

Cache HTTP

O cache HTTP armazena uma resposta associada a uma solicitação e reutiliza a resposta armazenada para solicitações subsequentes.

Há várias vantagens na reutilização. Primeiro, como não há necessidade de enviar a solicitação ao servidor de origem, quanto mais próximo o cliente estiver do cache, mais rápida será a resposta. O exemplo mais típico é quando o próprio navegador armazena um cache para solicitações do navegador.

Além disso, quando uma resposta pode ser reutilizada, o servidor de origem não precisa processar a solicitação — ou seja, não precisa analisar e encaminhar a solicitação, restaurar a sessão com base no cookie, consultar o banco de dados por resultados ou renderizar com o motor de templates. Isso reduz a carga no servidor.

O funcionamento correto do cache é fundamental para a saúde do sistema.

Tipos de cache

Na especificação de Cache HTTP, há dois tipos principais de cache: **caches privados** e **caches compartilhados**.

Cache privado

Um cache privado é um cache vinculado a um cliente específico — tipicamente um cache de navegador. Como a resposta armazenada não é compartilhada com outros clientes, um cache privado pode armazenar uma resposta personalizada para esse usuário.

Por outro lado, se conteúdos personalizados forem armazenados em um cache que não seja privado, outros usuários podem recuperar esses conteúdos — o que pode causar vazamento de informações de forma não intencional.

[Link da especificação RFC9111](#)

Se uma resposta contiver conteúdo personalizado e você quiser armazená-la apenas no cache privado, deve-se especificar a diretiva `private`.

Conteúdos personalizados geralmente são controlados por cookies, mas a presença de um cookie não indica necessariamente que a resposta é privada. Portanto, um cookie sozinho não torna a resposta privada.

Cache compartilhado

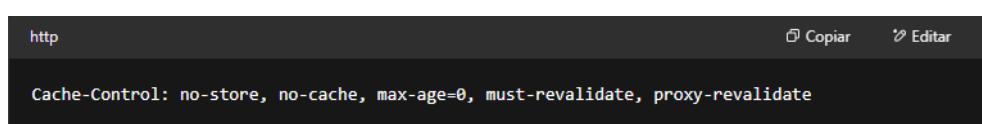
O cache compartilhado está localizado entre o cliente e o servidor, e pode armazenar respostas que podem ser compartilhadas entre usuários. Os caches compartilhados podem ser classificados em dois subtipos: **caches de proxy** e **caches gerenciados**.

Caches de proxy

Além da função de controle de acesso, alguns proxies implementam cache para reduzir o tráfego de saída da rede. Isso geralmente não é gerenciado pelo desenvolvedor do serviço, então deve ser controlado por meio de cabeçalhos HTTP apropriados, entre outros.

Contudo, no passado, implementações de cache de proxy desatualizadas — que não compreendiam adequadamente o padrão de cache HTTP — frequentemente causavam problemas aos desenvolvedores.

Cabeçalhos como o abaixo eram usados como forma de contornar essas implementações obsoletas de proxy-cache que não entendiam diretivas como `no-store`.



No entanto, nos últimos anos, com a popularização do HTTPS e a criptografia da comunicação cliente-servidor, os caches de proxy no caminho apenas fazem o túnel da resposta, e não se comportam como cache, na maioria dos casos. Assim, nesses cenários, não é necessário se preocupar com proxies-cache desatualizados que nem mesmo podem ver a resposta.

Caches gerenciados

Caches gerenciados são implantados explicitamente pelos desenvolvedores de serviço para aliviar o servidor de origem e entregar conteúdo de forma eficiente. Exemplos incluem **proxies reversos**, **CDNs** e **service workers** em combinação com a **Cache API**.

As características dos caches gerenciados variam conforme o produto utilizado. Em geral, você pode controlar o comportamento do cache por meio do cabeçalho Cache-Control e de seus próprios arquivos de configuração ou painéis administrativos.

Por exemplo, a especificação HTTP de cache não define uma forma explícita de deletar um cache — mas com um cache gerenciado, a resposta armazenada pode ser excluída a qualquer momento por meio de operações no painel, chamadas à API, reinicializações, etc. Isso permite uma estratégia de cache mais proativa.

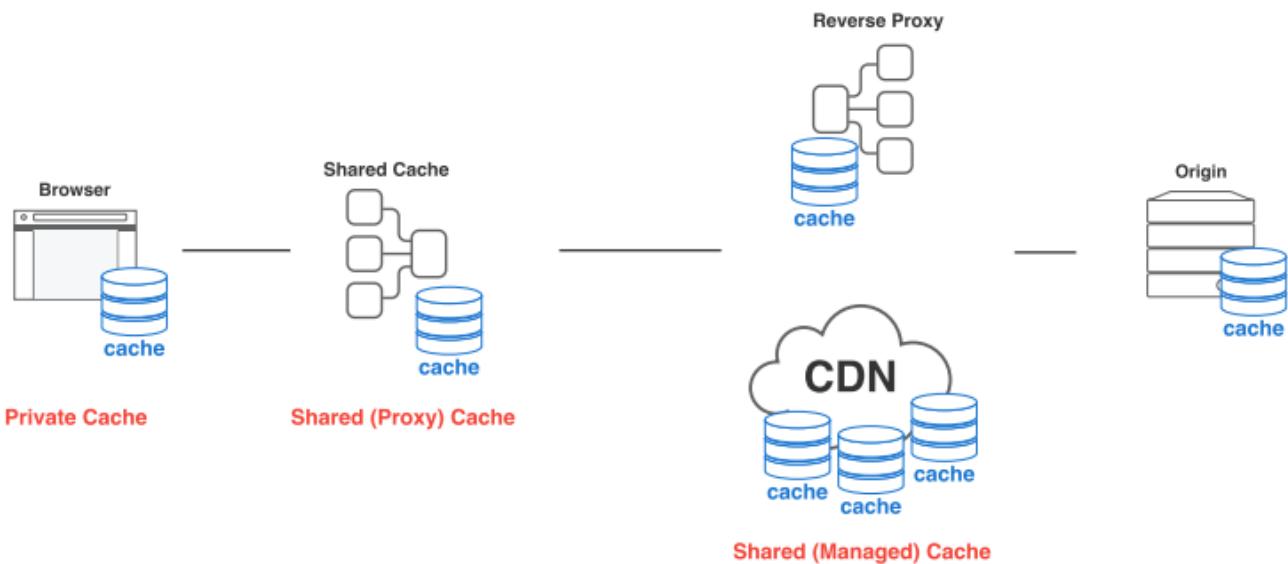
Também é possível ignorar os protocolos padrões do HTTP em favor de uma manipulação explícita. Por exemplo, é possível optar por não utilizar cache privado ou proxy, armazenando somente no cache gerenciado com controle próprio.

Exemplo com no-store:



O Varnish Cache usa a linguagem VCL (Varnish Configuration Language, um tipo de DSL) para lidar com o armazenamento em cache, enquanto os service workers combinados com a Cache API permitem que você crie essa lógica usando JavaScript. Isso significa que, se um cache gerenciado intencionalmente ignorar uma diretiva no-store, não há necessidade de considerá-lo como "não compatível" com o padrão. O que você deve fazer é evitar o uso de cabeçalhos "kitchen-sink" (com várias diretivas redundantes), ler cuidadosamente a documentação do mecanismo de cache gerenciado que você está usando e garantir que está controlando o cache corretamente pelas formas fornecidas por esse mecanismo.

Observe que alguns CDNs fornecem seus próprios cabeçalhos que são eficazes apenas para aquele CDN (por exemplo, Surrogate-Control). Atualmente, está em andamento o trabalho para definir o cabeçalho CDN-Cache-Control para padronizar isso.



Cache heurístico

O HTTP foi projetado para armazenar em cache o máximo possível, então, mesmo que nenhum Cache-Control seja fornecido, as respostas podem ser armazenadas e reutilizadas se certas condições forem atendidas. Isso é chamado de **cache heurístico**.

Por exemplo, veja a seguinte resposta. Essa resposta foi atualizada pela última vez há 1 ano:

```

h
Copiar Editar

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1024
Date: Tue, 22 Feb 2022 22:22:22 GMT
Last-Modified: Tue, 22 Feb 2021 22:22:22 GMT
<!doctype html>
...

```

Heuristicamente, sabe-se que o conteúdo que não foi atualizado por um ano inteiro provavelmente não será atualizado por algum tempo depois disso. Portanto, o cliente armazena essa resposta (apesar da ausência de max-age) e a reutiliza por um tempo. O tempo de reutilização depende da implementação, mas a especificação recomenda cerca de 10% (neste caso, 0,1 ano) do tempo desde que foi armazenada.

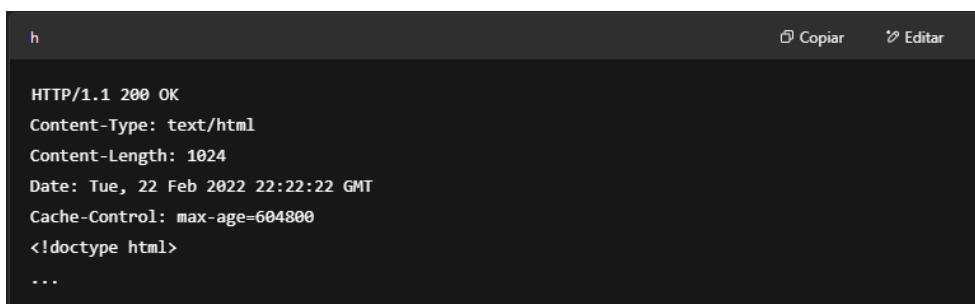
O cache heurístico é uma solução alternativa que surgiu antes da adoção ampla do Cache-Control, e basicamente todas as respostas deveriam especificar explicitamente esse cabeçalho.

Fresco e expirado com base na idade

Respostas HTTP armazenadas têm dois estados: **fresco** (*fresh*) e **expirado** (*stale*). O estado fresco geralmente indica que a resposta ainda é válida e pode ser reutilizada, enquanto o estado expirado significa que a resposta em cache já ultrapassou seu tempo de validade.

O critério para determinar quando uma resposta está fresca ou expirada é a **idade** (*age*). Em HTTP, "idade" é o tempo decorrido desde que a resposta foi gerada. Isso é similar ao **TTL** (time-to-live) em outros mecanismos de cache.

Considere o exemplo de resposta a seguir, onde 604800 segundos é igual a uma semana:



```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1024
Date: Tue, 22 Feb 2022 22:22:22 GMT
Cache-Control: max-age=604800
<!doctype html>
...
```

O cache que armazenou essa resposta calcula o tempo decorrido desde a geração da resposta e usa esse valor como a idade da resposta.

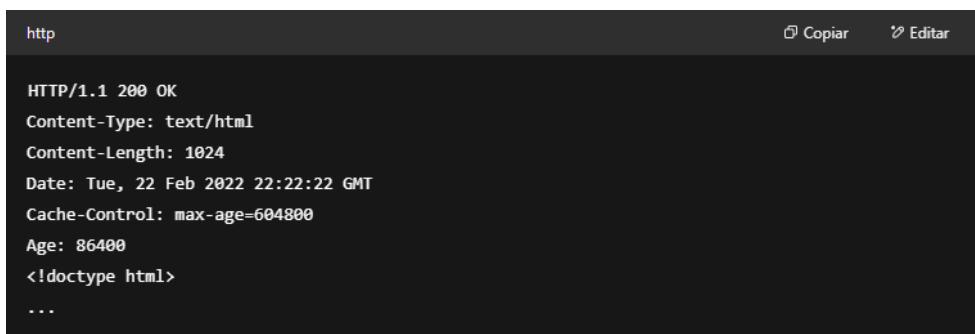
Para essa resposta, o significado de max-age é o seguinte:

- Se a idade da resposta for inferior a uma semana, a resposta está fresca.
- Se a idade da resposta for superior a uma semana, a resposta está expirada.

Enquanto a resposta armazenada continuar fresca, ela será usada para atender solicitações do cliente.

Se uma resposta estiver em um cache compartilhado, é possível informar ao cliente a idade da resposta.

Continuando o exemplo, se o cache compartilhado armazenou a resposta por um dia, ele enviaria a seguinte resposta para solicitações subsequentes:



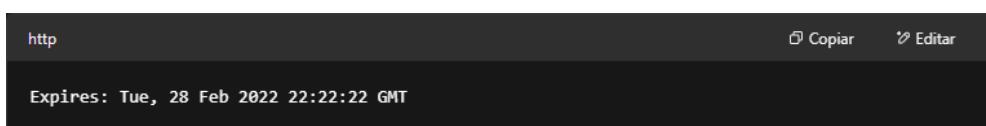
```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1024
Date: Tue, 22 Feb 2022 22:22:22 GMT
Cache-Control: max-age=604800
Age: 86400
<!doctype html>
...
```

O cliente que receber essa resposta a considerará fresca pelos **518.400 segundos** restantes — ou seja, a diferença entre max-age e Age.

Expires ou max-age

No HTTP/1.0, a validade era especificada pelo cabeçalho Expires.

O Expires especifica o tempo de vida do cache usando um horário explícito, em vez de um tempo decorrido.



```
Expires: Tue, 28 Feb 2022 22:22:22 GMT
```

No entanto, o formato de tempo é difícil de analisar, muitos bugs de implementação foram encontrados, e é possível causar problemas intencionando alterar o relógio do sistema. Por isso, max-age — que especifica um tempo decorrido — foi adotado para o Cache-Control no HTTP/1.1.

Se Expires e Cache-Control: max-age estiverem ambos disponíveis, max-age é preferido. Assim, não é necessário usar Expires agora que o HTTP/1.1 é amplamente utilizado.

Vary

A forma como respostas são diferenciadas entre si baseia-se essencialmente nas URLs:

URL	Corpo da resposta
https://example.com/index.html	<!doctype html>...
https://example.com/style.css	body { ... }
https://example.com/script.js	function main() { ... }

Mas o conteúdo das respostas nem sempre é o mesmo, mesmo que tenham a mesma URL. Especialmente quando há negociação de conteúdo, a resposta do servidor pode depender dos valores dos cabeçalhos de solicitação Accept, Accept-Language e Accept-Encoding.

Por exemplo, se uma resposta em inglês for retornada com o cabeçalho Accept-Language: en e for armazenada em cache, é indesejável reutilizá-la para uma solicitação com Accept-Language: ja. Nesse caso, pode-se instruir o cache a armazenar respostas separadas — com base no idioma — adicionando Accept-Language ao valor do cabeçalho Vary.

http	Copiar	Editar
Vary: Accept-Language		

Isso faz com que o cache use como chave uma combinação da URL da resposta e do cabeçalho Accept-Language da solicitação — em vez de usar apenas a URL.

URL	Accept-Language	Corpo da resposta
https://example.com/index.html	ja-JP	<!doctype html>...
https://example.com/index.html	en-US	<!doctype html>...

Se você estiver otimizando conteúdo (ex: design responsivo) com base no agente do usuário, pode haver a tentação de incluir User-Agent em Vary. Porém, o User-Agent tem muitas variações, o que reduz drasticamente a reutilização do cache. Portanto, prefira variar o comportamento com base em **detecção de recursos** em vez do User-Agent.

Se sua aplicação usa cookies para evitar que outros reutilizem conteúdos personalizados, é melhor especificar Cache-Control: private em vez de usar cookies no Vary.

Validação

Respostas expiradas não são descartadas imediatamente. O HTTP possui um mecanismo para transformar uma resposta expirada em uma resposta fresca ao consultar o servidor de origem. Isso é chamado de **validação**, ou às vezes **revalidação**.

A validação é feita por meio de uma requisição condicional que inclui os cabeçalhos If-Modified-Since ou If-None-Match.

If-Modified-Since

A seguinte resposta foi gerada às 22:22:22 e tem max-age de 1 hora, então sabemos que é válida até 23:22:22:

```
h Copiar Editar

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1024
Date: Tue, 22 Feb 2022 22:22:22 GMT
Last-Modified: Tue, 22 Feb 2022 22:00:00 GMT
Cache-Control: max-age=3600
<!doctype html>
...
```

Às 23:22:22, a resposta se torna expirada e não pode ser reutilizada. Então, o cliente envia uma requisição com o cabeçalho If-Modified-Since para perguntar ao servidor se houve alguma alteração desde o horário especificado:

```
http Copiar Editar

GET /index.html HTTP/1.1
Host: example.com
Accept: text/html
If-Modified-Since: Tue, 22 Feb 2022 22:00:00 GMT
```

O servidor responderá com 304 Not Modified se o conteúdo não tiver sido alterado desde o horário especificado:

```
http Copiar Editar

HTTP/1.1 304 Not Modified
Content-Type: text/html
Date: Tue, 22 Feb 2022 23:22:22 GMT
Last-Modified: Tue, 22 Feb 2022 22:00:00 GMT
Cache-Control: max-age=3600
```

Como essa resposta indica apenas "sem alteração", não há corpo de resposta — apenas um código de status — então o tamanho da transferência é extremamente pequeno. Ao recebê-la, o cliente revalida e torna a resposta armazenada novamente fresca, podendo reutilizá-la por mais 1 hora.

O servidor pode obter a data de modificação do sistema de arquivos do sistema operacional, o que é relativamente simples no caso de arquivos estáticos. No entanto, há problemas como o formato de data complexo e a dificuldade de sincronização de datas em servidores distribuídos.

Para resolver esses problemas, foi padronizado o uso do cabeçalho de resposta ETag como alternativa.

ETag / If-None-Match

O valor do cabeçalho de resposta ETag é um valor arbitrário gerado pelo servidor. Não há restrições sobre como o servidor deve gerar esse valor, portanto, ele pode basear-se em qualquer critério — como um hash do conteúdo ou um número de versão.

Por exemplo, se for usado um hash do conteúdo do recurso index.html, e o valor for "33a64df5", a resposta será:

```
http
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1024
Date: Tue, 22 Feb 2022 22:22:22 GMT
ETag: "33a64df5"
Cache-Control: max-age=3600
<!doctype html>
...
```

Se essa resposta se tornar expirada, o cliente usa o valor de ETag armazenado e o envia no cabeçalho If-None-Match para perguntar ao servidor se o recurso foi alterado:

```
http
GET /index.html HTTP/1.1
Host: example.com
Accept: text/html
If-None-Match: "33a64df5"
```

O servidor retornará 304 Not Modified se o valor de ETag for o mesmo. Se o recurso tiver mudado, o servidor retornará 200 OK com a nova versão.

Mas se o servidor determinar que o recurso solicitado deve ter um valor ETag diferente, o servidor responderá com 200 OK e a versão mais recente do recurso.

Observação: a RFC9110 prefere que os servidores enviem ETag e Last-Modified para uma resposta 200, se possível. Durante a revalidação do cache, se If-Modified-Since e If-None-Match estiverem presentes, If-None-Match terá precedência para o validador. Se você estiver considerando apenas o cache, pode achar que Last-Modified é desnecessário. No entanto, Last-Modified não é útil apenas para cache; é um cabeçalho HTTP padrão que também é usado por sistemas de gerenciamento de conteúdo (CMS) para exibir o horário da última modificação, por rastreadores para ajustar a frequência de rastreamento e para outros propósitos diversos. Portanto, considerando o ecossistema HTTP geral, é melhor fornecer ETag e Last-Modified.

Revalidação forçada

Se você não quiser que uma resposta seja reutilizada automaticamente e preferir sempre obter a versão mais recente do servidor, pode usar a diretiva no-cache para **forçar a validação**.

Ao adicionar Cache-Control: no-cache à resposta, juntamente com Last-Modified e ETag, o cliente receberá 200 OK se o conteúdo tiver sido atualizado, ou 304 Not Modified se não houver alterações.

Nota: O RFC9110 recomenda que os servidores enviem tanto ETag quanto Last-Modified em respostas 200, se possível. Durante a revalidação de cache, se If-Modified-Since e If-None-Match estiverem presentes, If-None-Match tem prioridade. Embora se pense que Last-Modified é desnecessário para cache, ele também é utilizado por CMS, crawlers e outros fins — portanto, é melhor fornecer ambos.

Exemplo:

```
http
Cache-Control: no-cache
Last-Modified: Tue, 22 Feb 2022 22:00:00 GMT
ETag: "deadbeef"
```

É comum dizer que a combinação max-age=0 e must-revalidate tem o mesmo efeito que no-cache. De fato:

- max-age=0: torna a resposta imediatamente expirada

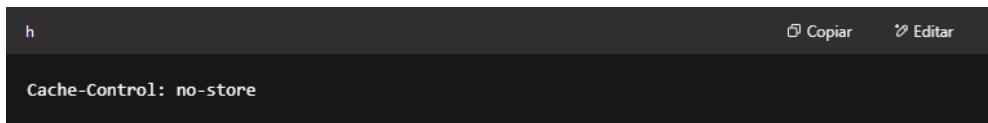
- must-revalidate: exige que a resposta não seja reutilizada sem validação

Essa combinação era uma solução para servidores antigos que não entendiam no-cache. Hoje, com servidores compatíveis com HTTP/1.1, o recomendado é simplesmente usar no-cache.

Não armazenar no cache

A diretiva no-cache **não impede o armazenamento** da resposta — ela apenas impede a **reutilização sem validação**.

Se você **não quiser que a resposta seja armazenada em nenhum cache**, use no-store.



No entanto, na prática, um requisito de "não armazenar em cache" geralmente significa:

- Não permitir que outros além do cliente específico armazenem a resposta, por questões de privacidade;
- Garantir sempre informações atualizadas;
- Evitar comportamentos inesperados em implementações antigas.

Nesses casos, no-store pode não ser a diretiva mais apropriada.

Não compartilhar com outros

Se uma resposta com conteúdo personalizado for visível para outros usuários do cache, isso é um problema.

A diretiva private garante que a resposta será armazenada apenas para aquele cliente específico, evitando vazamentos:



Mesmo se no-store for usado, private também deve ser incluído.

Fornecer conteúdo atualizado sempre

A diretiva no-store impede que a resposta seja armazenada, **mas não apaga** respostas já armazenadas para a mesma URL.

Ou seja, se já houver uma resposta antiga armazenada para uma URL, o uso de no-store não impedirá sua reutilização. Já no-cache força o cliente a enviar uma solicitação de validação antes de reutilizar qualquer resposta armazenada:



Se o servidor **não suportar requisições condicionais**, é possível forçar o cliente a acessar o servidor sempre, recebendo uma nova resposta 200 OK.

Lidando com implementações antigas

Como solução alternativa para implementações antigas que ignoram no-store, pode-se ver o uso de cabeçalhos "kitchen-sink", como o exemplo abaixo:

```
http Copiar Editar  
Cache-Control: no-store, no-cache, max-age=0, must-revalidate, proxy-revalidate
```

É recomendado usar no-cache como alternativa para lidar com essas implementações, pois o servidor sempre receberá a solicitação.

Se a preocupação for com o **cache compartilhado**, adicione também private:

```
http Copiar Editar  
Cache-Control: no-cache, private
```

O que se perde ao usar no-store

Pode parecer que adicionar no-store é a forma correta de evitar cache. No entanto, **não é recomendado usar no-store de forma liberal**, pois isso faz com que você perca muitas vantagens que o HTTP e os navegadores oferecem — como o cache de navegação (voltar/avançar).

Portanto, para aproveitar ao máximo os recursos da plataforma web, prefira usar:

```
http Copiar Editar  
Cache-Control: no-cache, private
```

Recarregar e forçar recarregar

A validação pode ser feita **não apenas nas respostas**, mas também **nas requisições**.

Recarregar (Reload)

Para recuperar a janela de um estado corrompido ou atualizar para a versão mais recente de um recurso, os navegadores oferecem a função de "recarregar".

Uma visão simplificada da requisição HTTP feita durante um reload:

```
http Copiar Editar  
GET / HTTP/1.1  
Host: example.com  
Cache-Control: max-age=0  
If-None-Match: "deadbeef"  
If-Modified-Since: Tue, 22 Feb 2022 20:20:20 GMT
```

O max-age=0 na requisição indica "reutilizar respostas com idade 0 ou menos", o que na prática **evita reutilizar o cache intermediário**. Isso resulta em uma requisição validada por If-None-Match e If-Modified-Since.

Esse comportamento também é definido no padrão [Fetch](#), e pode ser reproduzido em JavaScript com:

```
js Copiar Editar  
fetch("/", { cache: "no-cache" });
```

⚠️ Nota: "reload" **não** é o modo correto para um reload normal; o correto é "no-cache".

Forçar recarregar (Force Reload)

Durante um recarregamento forçado (usado para contornar cache), a requisição se parece com:

```
http Copiar Editar  
GET / HTTP/1.1  
Host: example.com  
Pragma: no-cache  
Cache-Control: no-cache
```

Essa requisição **não é condicional**, então o cliente garante que receberá um 200 OK do servidor.

Também pode ser reproduzido via JavaScript com:

```
js Copiar Editar  
fetch("/", { cache: "reload" });
```

⚠ Aqui sim, "reload" é o modo correto para um **force reload**.

Evitando revalidação

Conteúdo que **nunca muda** deve receber um max-age longo, com uso de **cache busting** — como incluir número de versão ou hash no URL.

No entanto, mesmo quando o conteúdo é imutável, ao recarregar a página, o navegador pode enviar uma requisição de revalidação.

Para evitar isso, use a diretiva immutable:

```
http Copiar Editar  
Cache-Control: max-age=31536000, immutable
```

Isso evita revalidações desnecessárias em recarregamentos.

Obs.: Em vez de implementar immutable, o navegador Chrome modificou sua implementação para **não revalidar sub-recursos durante reloads**.

Deletando respostas armazenadas

Não há uma forma padrão de deletar respostas armazenadas com max-age longo em servidores intermediários.

Exemplo: a seguinte resposta foi armazenada:

```
http Copiar Editar  
HTTP/1.1 200 OK  
Content-Type: text/html  
Content-Length: 1024  
Cache-Control: max-age=31536000  
<!doctype html>  
...
```

Mesmo se a resposta for modificada no servidor, ela **não será substituída** enquanto não expirar, pois nenhuma nova requisição chegará ao servidor.

Uma alternativa mencionada na especificação é fazer uma requisição para a mesma URL com um método inseguro como POST, mas isso é inviável em muitos clientes.

Limpando cache do navegador

O cabeçalho `Clear-Site-Data`: cache pode limpar caches do navegador, mas **não afeta caches intermediários**. Do contrário, as respostas permanecerão armazenadas até a expiração de `max-age`, a não ser que o usuário recarregue, force-recarregue ou limpe o histórico.

http Copiar Editar

```
Clear-Site-Data: "cache"
```

Quando o cache reduz o número de acessos ao servidor, o servidor **perde controle sobre aquela URL**. Se quiser manter esse controle (por exemplo, se o recurso for frequentemente atualizado), adicione:

http Copiar Editar

Cache-Control: no-cache

Assim, o servidor sempre receberá a solicitação.

Colapso de requisições (Request collapse)

O cache compartilhado geralmente está localizado **antes do servidor de origem** e serve para reduzir o tráfego.

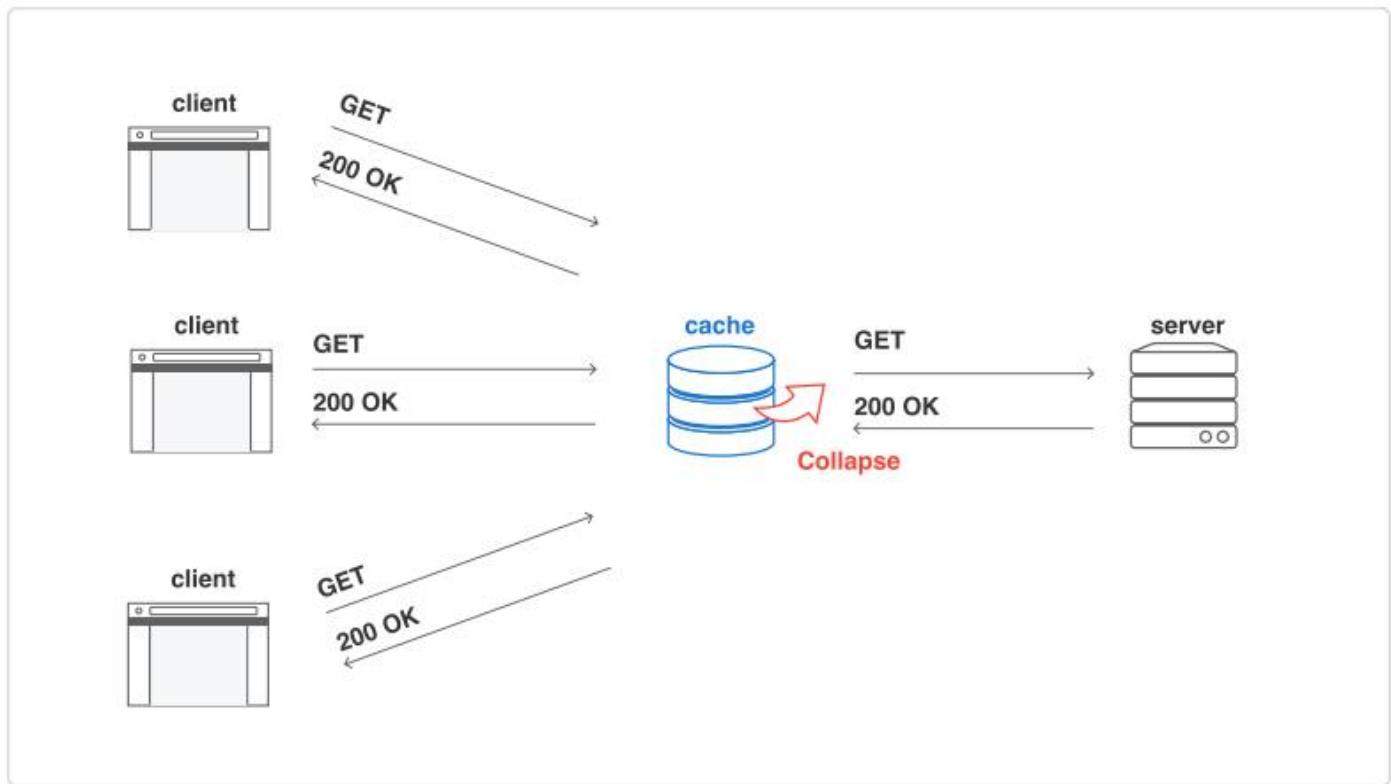
Quando múltiplas requisições idênticas chegam ao mesmo tempo, o cache intermediário encaminha **uma única requisição** ao servidor e reutiliza o resultado para todos os clientes. Isso é chamado de **request collapse**.

Mesmo que a resposta tenha no-cache ou max-age=0, ela poderá ser reutilizada nesse contexto.

Se a resposta for personalizada e não quiser que ela seja compartilhada durante o colapso, adicione:

[http](#) Copiar Editar

Cache-Control: private



Padrões comuns de cache

Há muitas diretrivas na especificação de Cache-Control, e entender todas pode ser difícil. No entanto, a maioria dos sites pode ser coberta por uma **combinação de poucos padrões**.

Esta seção descreve os padrões mais comuns no design de caches.

Configurações padrão

Como mencionado anteriormente, o comportamento padrão de cache (isto é, quando a resposta **não** contém Cache-Control) **não é simplesmente "não armazenar"**, mas sim aplicar o chamado **cache heurístico**.

Para evitar o cache heurístico, é **preferível fornecer explicitamente um cabeçalho Cache-Control para todas as respostas**.

Para garantir que a versão mais recente dos recursos seja sempre entregue, é comum definir:

```
http
Copiar Editar
Cache-Control: no-cache
```

Além disso, se o serviço utiliza cookies ou métodos de login, e o conteúdo é personalizado para cada usuário, também deve ser usado:

```
http
Copiar Editar
Cache-Control: no-cache, private
```

Cache Busting

Os recursos que funcionam melhor com cache são **arquivos estáticos e imutáveis**, cujo conteúdo nunca muda. Para recursos que mudam, é prática comum **alterar a URL sempre que o conteúdo for alterado**, de forma que cada versão possa ser armazenada em cache por mais tempo.

Considere o HTML:

```
html Copiar Editar

<script src="bundle.js"></script>
<link rel="stylesheet" href="build.css" />
<body>
  hello
</body>
```

Em desenvolvimento moderno, arquivos JavaScript e CSS são frequentemente atualizados. Se as versões desses arquivos estiverem fora de sincronia, a exibição do site pode quebrar.

Assim, esse HTML dificulta o cache de bundle.js e build.css com max-age.

A solução é incluir um **componente variável** (como versão ou hash) na URL:

```
bash Copiar Editar

# versão no nome do arquivo
bundle.v123.js

# versão na query string
bundle.js?v=123

# hash no nome do arquivo
bundle.YsAIAAA-QG4G6kCMAMBAAAAAAoK.js

# hash na query string
bundle.js?v=YsAIAAA-QG4G6kCMAMBAAAAAAoK
```

Como o cache diferencia recursos com base na URL, **uma URL alterada não reutilizará o cache anterior**.

Isso permite armazenar os arquivos JavaScript e CSS por muito tempo. Mas por quanto tempo?

A especificação **QPACK** (para compressão de cabeçalhos no HTTP/3) sugere valores otimizados. Veja:

Código	Cache-Control
36	max-age=0
37	max-age=604800 (1 semana)
38	max-age=2592000 (1 mês)
39	no-cache
40	no-store
41	public, max-age=31536000 (1 ano)

Ao usar esses valores, o cabeçalho pode ser comprimido em **1 byte** no HTTP/3.

⚠️ Nota: o valor 41 é o mais longo, mas com public. Isso permite cache mesmo com cabeçalho Authorization. Se a resposta for personalizada, isso pode causar problemas. Prefira o valor 38 (1 mês) nesses casos.

Validação

Não esqueça de definir os cabeçalhos Last-Modified e ETag para evitar retransferência de recursos em recarregamentos.

Para arquivos estáticos pré-construídos, é fácil gerar esses cabeçalhos:

```
http
Copiar Editar

Last-Modified: Tue, 22 Feb 2022 20:20:20 GMT
ETag: "YsAIAAAA-QG4G6kCMAMBAAAAAAoK"
```

E a resposta completa:

```
http
Copiar Editar

HTTP/1.1 200 OK
Content-Type: text/javascript
Content-Length: 1024
Cache-Control: public, max-age=31536000, immutable
Last-Modified: Tue, 22 Feb 2022 20:20:20 GMT
ETag: "YsAIAAAA-QG4G6kCMAMBAAAAAAoK"
```

⚠ A diretiva public só deve ser usada se for necessário permitir armazenamento em cache com cabeçalho Authorization. Caso contrário, não é exigida.

Recursos principais

Ao contrário de sub-recursos (JS, CSS, imagens), os **recursos principais** (como HTML) **não podem usar cache busting**, pois suas URLs geralmente **não podem ser modificadas** com versões.

Se o HTML for armazenado, a **versão mais recente pode não ser exibida**, mesmo que o conteúdo tenha mudado no servidor.

Nesse caso, use no-cache (em vez de no-store), pois não queremos evitar o cache, apenas garantir que ele esteja atualizado.

Também é útil fornecer Last-Modified e ETag, para permitir requisições condicionais com resposta 304 Not Modified, se o conteúdo não tiver mudado:

```
http
Copiar Editar

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1024
Cache-Control: no-cache
Last-Modified: Tue, 22 Feb 2022 20:20:20 GMT
ETag: "AAPuIbAOdvAGEETbgAAAAAAABAAE"
```

Se o HTML for personalizado com cookies (por exemplo, após login), adicione:

```
http
Copiar Editar

Cache-Control: no-cache, private
Set-Cookie: __Host-SID=AHNtAyt3fvJrUL5g5tnGwER; Secure; Path=/; HttpOnly
```

Sub-recursos e outros

O mesmo se aplica a arquivos como:

- favicon.ico
- manifest.json
- .well-known
- Endpoints de API cujas URLs não podem ser alteradas

Quase todo conteúdo da web pode ser tratado por uma **combinação dos dois padrões** apresentados:

1. Cache busting para sub-recursos (com max-age, immutable)
 2. no-cache para recursos principais, com Last-Modified, ETag, e private se necessário
-

Mais sobre caches gerenciados

Com as estratégias acima, sub-recursos podem ser armazenados por longos períodos. Já **recursos principais** não podem, pois não há como deletar explicitamente os conteúdos em cache com a especificação padrão.

Entretanto, é possível com o uso de **cache gerenciado**, como:

- **CDNs** (com API ou painel para limpar cache)
- **Service Workers** (que podem deletar a Cache API após atualização)

Para mais informações, veja a documentação do seu CDN ou leia sobre [Service Worker API](#).

Veja também

- [RFC 9111: HTTP/1.1 – Caching](#)
- [Caching Tutorial – Mark Nottingham](#)